LA BLOCKCHAIN : SMART CONTRACT

Christophe Ozcan

christophe.ozcan@esme.fr

Blockchain 2.0 : ETHEREUM & SMART CONTRACT

Bitcoin la première générération

Introduction à Ethereum la seconde génération

Smart contract définition et concept

Eléments et caractéristiques des Smart contract

Solidity

ABI Smart Contract

IDE Remix

Bitcoin la première génération

- Cryptodevise : bitcoin
- Registre public et distribué
- Immuabilité des transactions
- Réseau P2P ouvert
- Utilisation du langage « scripting » pour réaliser les transactions et d'une structure de donnée « Arbre de Merkle »
- Algorithme cryptographique : Proof Of Work
- Principale fonction : Moyen de paiement

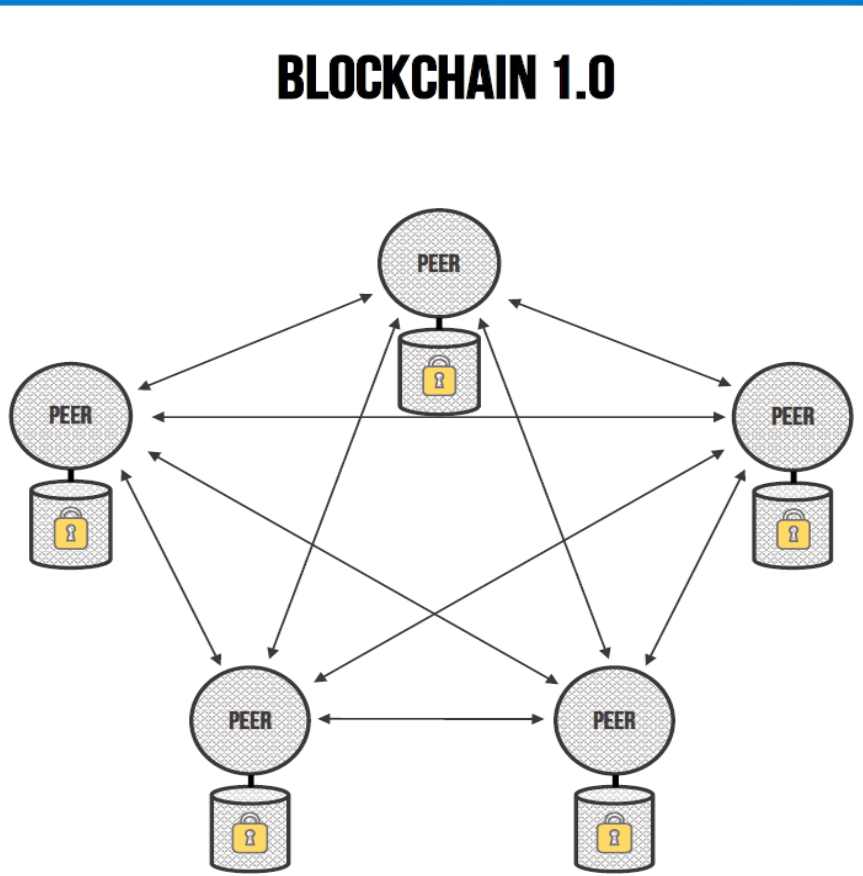
Blockchain 2.0 : Ethereum & Smart Contract

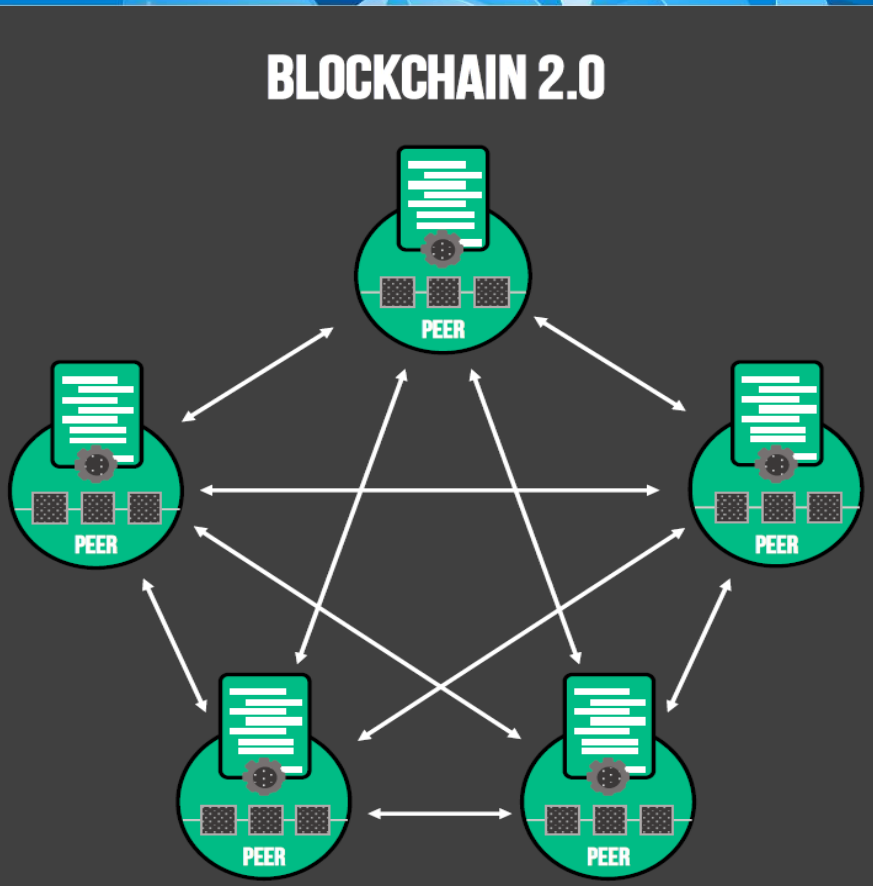
Introduction à Ethereum la seconde génération

- Cryptodevise : Ether
- Registre public et distribué
- Immuabilité des transactions
- Réseau P2P ouvert
- Utilisation du langage « Solidity » et d'une structure de donnée « Arbre de Merkle »
- Algorithme cryptographique : Proof Of Work
- Nouveauté : Smart contract

Blockchain 2.0 : Ethereum & Smart Contract

Introduction à Ethereum la seconde génération



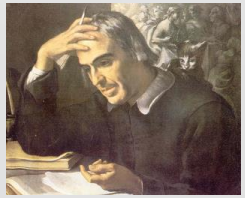


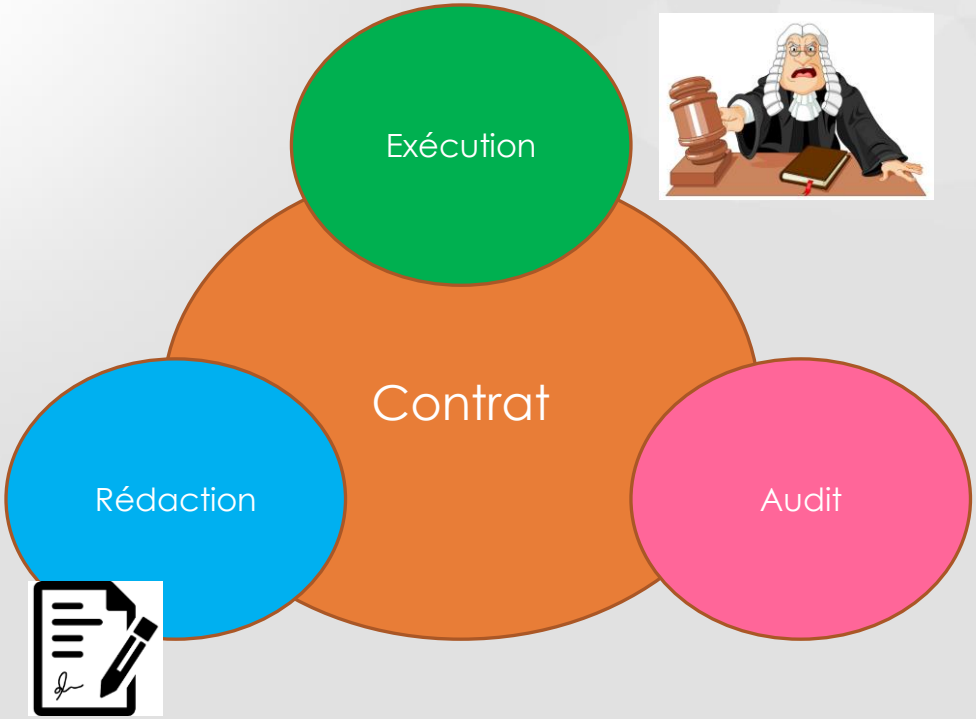
Vitalik Buterin, 22 ans, programmeur et inventeur d'Ethereum, l'a a expliqué lors d'un récent événement au DC Blockchain Summit qu'il s'agissait d'un approche de contrat intelligent.

Un actif ou une devise est transféré dans un programme informatique qui exécute ce code automatiquement lorsque les conditions d'exécutions sont réuni.

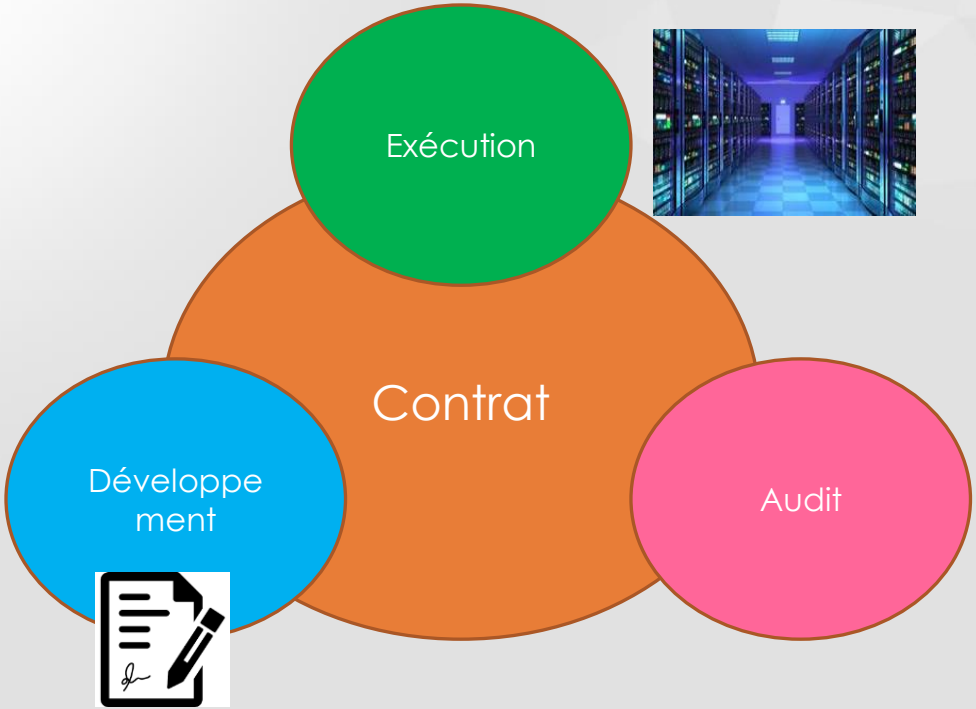
Dans le cas d'un transfert d'actif il est transféré automatiquement à un individu à l'autre, ou s'il doit être immédiatement remboursé à la personne qui l'a envoyé ou une combinaison de deux par exemple. Entre-temps, le registre décentralisé de la Blockchain stocke et reproduit le document qui lui confère une certaine sécurité et immuabilité.

- Traditionnel contrat



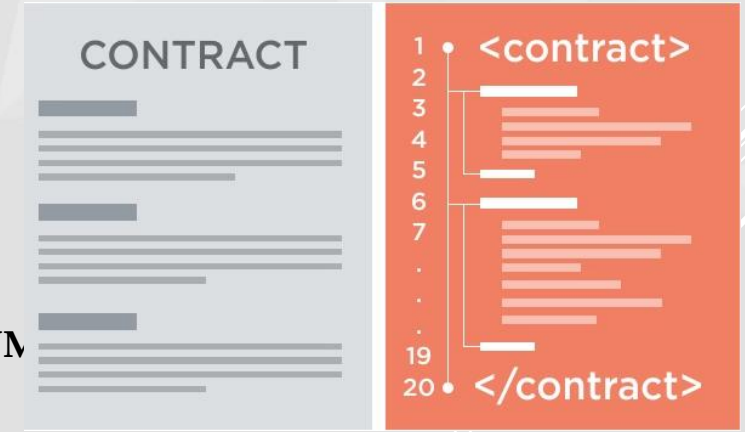


- Contrat numérique immuable



- Qu'est ce qu'un smart contract ?

C'est un terme utilisé pour désigner
(i) la transcription en langage informatique de conditions
(ii) enregistrées sur un registre décentralisé et
(iii) exécutées automatiquement
(iv) en fonction d'un ou plusieurs événements déclencheurs



- **Lignes de codes**
- **Exécutées automatiquement par une machine virtuelle (VM**
- **Evénement (s) déclencheur(s)**
- **Auditable**

La machine virtuelle Ethereum

La machine virtuelle Ethereum ou EVM est l'environnement d'exécution des contrats intelligents dans Ethereum.

Il est non seulement en bac à sable, mais en fait complètement isolé, ce qui signifie que le code exécuté dans l'EVM n'a pas accès au réseau, au système de fichiers ou à d'autres processus. Les contrats intelligents ont même un accès limité aux autres contrats intelligents.

Comptes

Il existe deux types de comptes dans Ethereum qui partagent le même espace d'adressage: les comptes externes contrôlés par des paires de clés publiques-privées (c'est-à-dire les humains) et les comptes contractuels contrôlés par le code stocké avec le compte.

L'adresse d'un compte externe est déterminée à partir de la clé publique tandis que l'adresse d'un contrat est déterminée au moment de la création du contrat.

Smart contract définition et concept

- Exemple

```
/* Allow another contract to spend some tokens in your behalf */
function approve(address _spender, uint256 _value)
    returns (bool success) {
    allowance[msg.sender][_spender] = _value;
    return true;
}

/* Approve and then comunicate the approved contract in a single tx */
function approveAndCall(address _spender, uint256 _value, bytes _extraData)
    returns (bool success) {
    tokenRecipient spender = tokenRecipient(_spender);
    if (approve(_spender, _value)) {
        spender.receiveApproval(msg.sender, _value, this, _extraData);
        return true;
    }
}

/* A contract attempts to get the coins */
function transferFrom(address _from, address _to, uint256 _value) returns (bool success) {
    if (balanceOf[_from] < _value) throw;                 // Check if the sender has enough
    if (balanceOf[_to] + _value < balanceOf[_to]) throw;  // Check for overflows
    if (_value > allowance[_from][msg.sender]) throw;   // Check allowance
    balanceOf[_from] -= _value;                           // Subtract from the sender
    balanceOf[_to] += _value;                             // Add the same to the recipient
    allowance[_from][msg.sender] -= _value;
    Transfer(_from, _to, _value);
    return true;
}

/* This unnamed function is called whenever someone tries to send ether to it */
function () {
    throw;     // Prevents accidental sending of ether
}
```

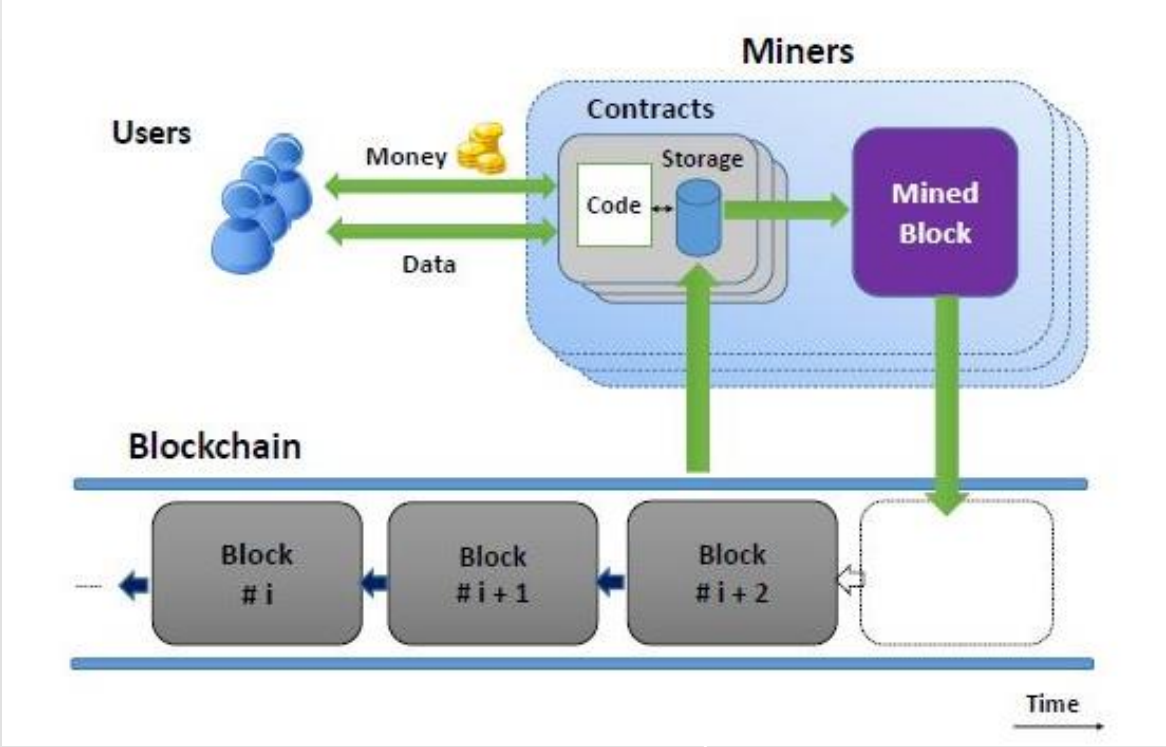
Eléments et caractéristiques des Smart contract

Deux types de transaction :
- Cryptodevise : Ether
- Smart contract : Code

Acteur supplémentaire :
- EVM : Ethereum Virtuel Machine

Nouveau langage Solidity :
- Orienté Object
- Compilateur

Exécution du code par l'EVM :
- Consommation de gaz « Ether »
- Auditable



- Qu'est ce que le gaz ?

Unité de mesure de consommation en ether (wei) de computation par les EVM (Ethereum Virtual Machine). A l'image du gasoil chaque transaction ou exécution de smart contract brule de la puissance de cacul.

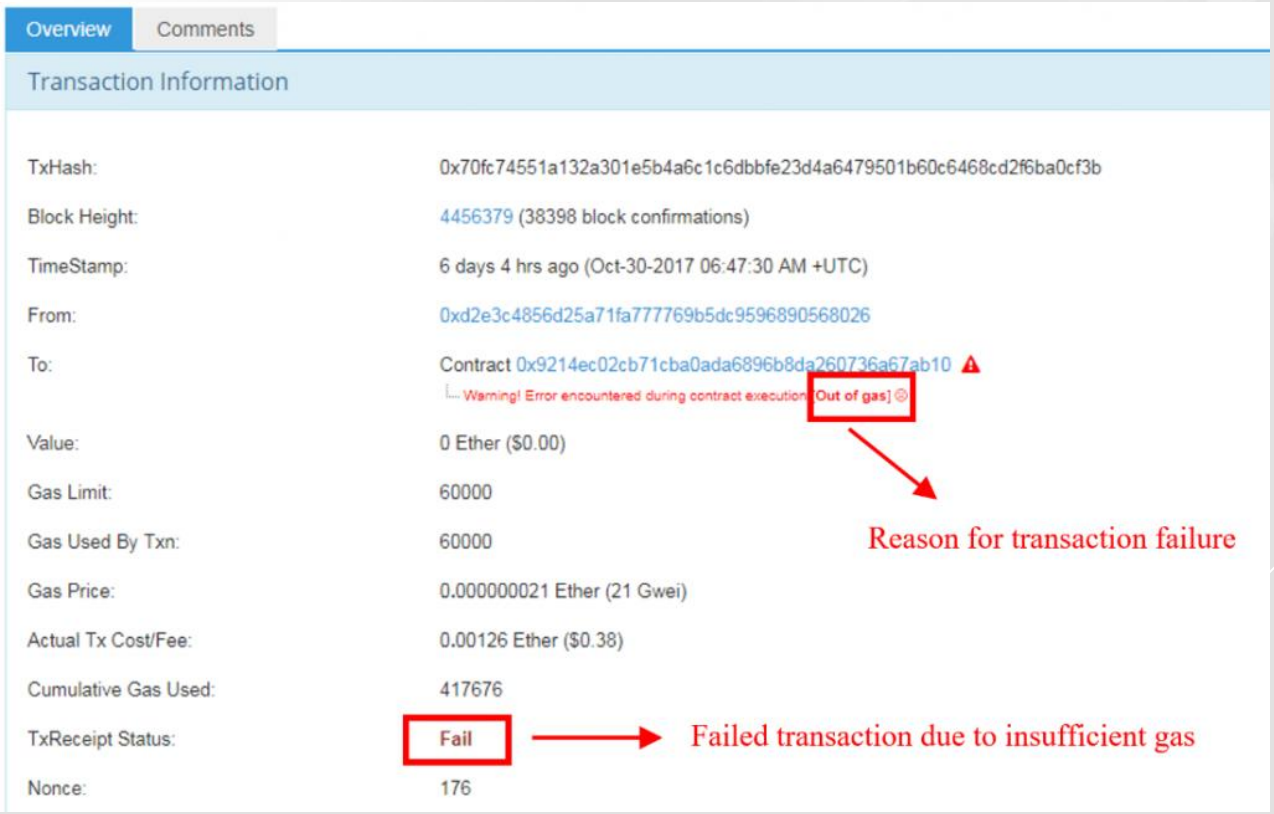
A l'image de la rémunération par les mineurs en ether qui sont rémunérés pour valider une transaction (fees) et déterminer le nouveau bloc (bloc reward).

Chaque opération a une dépense de gaz estimé lors du déploiement ou l'appel d'un smart contract. Plus un smart contract contient des opérations arithmétiques et des conditions d'exécutions (if, else ..) plus son estimation de côut en gaz est important.

Les unités de mesure de gaz

1000000000000000000	Wei
1000000000000000	Kwei
1000000000000	Mwei
1000000000	Gwei
1000000	Szabo
1000	Finney
1	Ether
0.001	Kether
0.000001	Methe
0.000000001	Gether
0.000000000001	Tether

- Qu'est ce que la limite de gaz en pratique ?



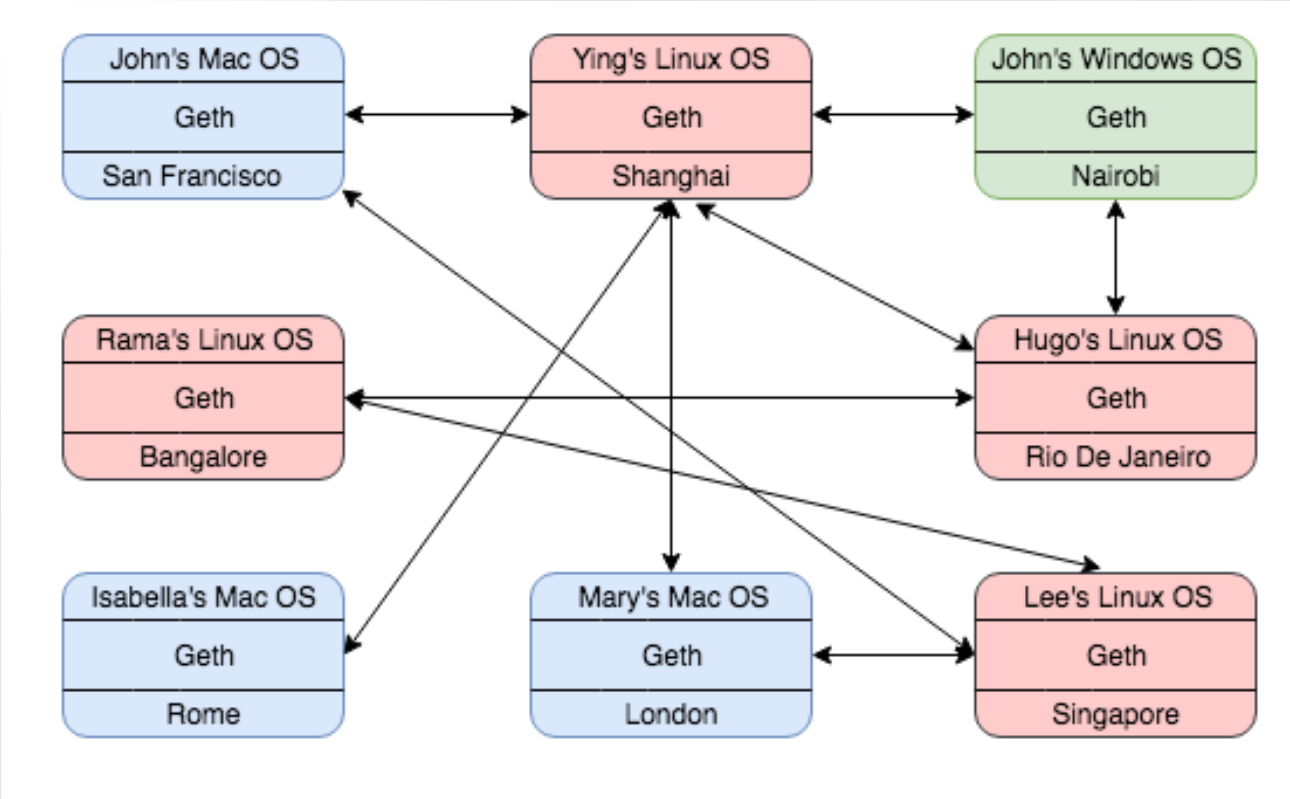
- Analyse d'une consommation de gaz par l'exécution d'un smart contract

Overview | Comments

Transaction Information

TxHash:	0x08b36b754691aa6f0608cb983bd23f2eec045a40f6ea41165dd48e8046af1514
TxReceipt Status:	Success
Block Height:	5082447 (23 block confirmations)
TimeStamp:	4 mins ago (Feb-13-2018 10:58:24 AM +UTC)
From:	0xdc7693bd416f4627871c82b4fc030e42238921b3
To:	0x27bd240886d755e1d273a21d2f00d8598c1c5724
Value:	1.01682595274441134 Ether ($846.17)
Gas Limit:	21000
Gas Used By Txn:	21000
Gas Price:	0.000000008 Ether (8 Gwei)
Actual Tx Cost/Fee:	0.000168 Ether ($0.14)
Cumulative Gas Used:	866792
Nonce:	0

- Le client Geth pour devenir un nœud du réseau



- Qu'est ce que le language Solidity ?

Solidity est le langage de programmation le plus utilisé pour écrire des contrats intelligents « smart contracts » à exécuter sur la blockchain Ethereum. C'est un langage de haut niveau qui, une fois compilé, est converti en code octet EVM (Ethereum Virtual Machine).

Ceci est très similaire au monde de Java où il y a des langages JVM comme Scala, Groovy, Clojure, JRuby etc. Tout cela sur la compilation génère un code octet qui s'exécute dans la JVM (Java Virtual Machine). Vous pouvez également créer un langage comme Solidity tant que vous suivez les spécifications et que votre langage compile jusqu'au code d'octet EVM valide!

Il y a aussi un très bon IDE basé sur un navigateur où vous pouvez écrire des contrats, les compiler et les déployer dans la blockchain ici: http://remix.ethereum.org/

- Documentation de Solidity : https://docs.soliditylang.org/en/v0.8.3/
- IDE REMIX : https://remix.ethereum.org/

Solidity

- Version du Solidity

```
1  pragma solidity ^0.6.12;
2
3  // SPDX-License-Identifier: GPL-3.0
```

- Librairie de smart contracts importés dans votre smart contract principale

```
5  import "./Ownable.sol";
6  import "./SafeMath.sol";
```

- Nom de votre Smart Contract et héritage

```
8  contract Election is Ownable {
9
```

- Déclarations des variables de type structure, uint et mapping

```
12      // Model a Candidate
13      struct Candidate {
14          uint256 id;
15          string name;
16          uint voteCount;
17      }
18
19      // Store accounts that have voted
20      mapping(address => bool) public voters;
21      // Store Candidates
22      // Fetch Candidate
23      mapping(uint => Candidate) public candidates;
24      // Store Candidates Count
25      uint public candidatesCount;
26
```

- Création d'un événement permettant de notifié le résultat sous forme de log

```
27      // voted event
28      event votedEvent ( uint indexed _candidateId);
29
```

- Définition d'une fonction de type internal ou external ayant une visibilité privé ou public.

```
30    function addCandidate (string memory _name) internal {
31        candidatesCount ++;
32        candidates[candidatesCount] = Candidate(candidatesCount, _name, 0);
33    }
34
35    function vote (uint _candidateId) public {
36        // require that they haven't voted before
37        require(!voters[msg.sender]);
38        // require a valid candidate
39        require(_candidateId > 0 && _candidateId <= candidatesCount);
40
41        // record that voter has voted
42        voters[msg.sender] = true;
43
44        // update candidate vote Count
45        candidates[_candidateId].voteCount ++;
46
47        // trigger voted event
48        emit votedEvent (_candidateId);
```

L'ABI de votre smart contract

Le Contract Application Binary Interface (ABI) est le moyen standard d'interagir avec les contrats dans l'écosystème Ethereum, à la fois de l'extérieur de la blockchain et pour l'interaction contrat à contrat. Les données sont codées en fonction de leur type, comme décrit dans la spécification officielle.

Le codage n'est pas auto-descriptif et nécessite donc un schéma pour le décoder.


```
[
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": true,
                "internalType": "address",
                "name": "previousOwner",
                "type": "address"
            },
            {
                "indexed": true,
                "internalType": "address",
                "name": "newOwner",
                "type": "address"
            }
        ],
        "name": "OwnershipTransferred",
        "type": "event"
    },
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": true,
                "internalType": "uint256",
                "name": "_candidateId",
                "type": "uint256"
            }
        ],
        "name": "votedEvent",
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
```

L'IDE REMIX